TOTA Mini-Tutorial

Beware. TOTA is a research project. The current implementation has been realized for the sole purpose of testing TOTA ideas, and verifying its feasibility. It is not a plug-and-play reliable toolbox.

TOTA distribution

The main TOTA distribution consists of the following sub-projects:

- **ConfigFile.** Contains a number of XML configuration files needed to start all the projects.
- **NetworkEmulator.** This is a simple network emulator allowing to create different kind of network infrastructures (manet, wired and satellite networks). The project contains classes to run the above networks both in a simulated environment, in a real network and in a mixed-mode environment in which some real devices are mapped into simulated nodes.
- **CorePeerServices.** This project contains a number of simple middleware infrastructures (e.g., event dispatcher, tuple space) that are used as basic building blocks for TOTA operations.
- **Lime.** This is a simple implementation of the Lime middleware. TOTA uses this middleware as a building block.
- **GPSRadar.** This project contains classes to simulate the behavior of a GPS and a radar localization mechanisms. Such kind of devices are used to enable advanced TOTA functionalities
- **TOTA.** This project contains the TOTA middleware and the TOTA tuples class hierarchy.

All the sub-projects can be started independently (once trivial classpath dependencies are resolved).

TOTA Getting Started

To start a new TOTA application, you can proceed with the following steps.

Create a new project and import all the class directories of the projects in the TOTA distribution.

Create the following agent application that uses basic TOTA functionalities. The agent is very simple: it creates a HopTuple and injects it in the network, then it moves and read the tuples in its local tuple space and in the tuple spaces of its one-hop neighborhood.

```
package simpleapplication;

import java.util.Vector;

import tota.middleware.TotaMiddleware;
import tota.tuples.TotaTuple;
import tota.tuples.hop.GradientTuple;
import tuplespace.tuples.Tuple;
import emulator.agent.AbstractAgent;
import emulator.peers.Peer;
import emulator.peers.PeerInterface_ApplicationSide;
import emulator.utils.GenPoint;

public class AppAgent extends AbstractAgent {

 private PeerInterface_ApplicationSide peer;
 private TotaMiddleware tota;


 /*
```

```
  * The application agent is created on a peer.
  * It creates a TOTA middleware
  */

 public AppAgent(PeerInterface_ApplicationSide peer) {
  this.peer = peer;
  tota = new TotaMiddleware(peer);
  System.out.println(peer.getAddress() + " starting...");
 }


 /*
  * The simulator ciclically invokes this step method with a time step value.
  * The first thing to do is to run the tota middleware (that is also animated in
  * a step-like way).
  */

 public void step(int time) {
   tota.step(time);

   /* since all the nodes run the same agent, it is possible to differentiate their behavior on the
basis of their name, or on the basis of the name of the peer in which they are running. */
   if(peer.toString().equals("P0")) {

       // inject a tuple
       if(time == 10) {
     TotaTuple t = new GradientTuple();
     t.setContent("<content=ciao>"); // this notation is mandatory
     tota.inject(t);
       }

       // move. The agent cannot move but can ask its peer to move
       if(time == 50) {
        Peer p = (Peer) peer.getRealIdentity();
        p.emu.move(p.toString(), new GenPoint(100, 100));
       }

       // read tuples in the local tuple space and in the neighbor tuple spaces.
       if(time == 100) {
        // this template tuple matches with all the tuples
        Tuple allTemplate = new Tuple("<content=*>");
        // read the local tuple space
        Vector local = tota.read(allTemplate);
        // read the remote tuple space
        Vector remote = tota.readOneHop(allTemplate);
        // print the results
        System.out.println("\n"+peer.toString()+" local tuples:");
        for(int i=0; i<local.size();i++)
         System.out.println(local.get(i));
        System.out.println("\n"+peer.toString()+" remote tuples:");
        for(int i=0; i<remote.size();i++)
         System.out.println(remote.get(i));
       }

   }
 }

 // interface method empty implementation
 public synchronized void showAgentFrame() {}
 public synchronized void hideAgentFrame() {peer.shutDown();}
 public synchronized Object getRealIdentity() {return this;}
}
```

Once the agent is ready, you have to create a simulation to run it.

```
package simpleapplication;

import emulator.utils.StaticUtilities;

public class Start {
 private static String fileName = "totaapp.xml";
 public static void main(String[] args) {
  StaticUtilities.startSimulation(StaticUtilities.EMULATOR,fileName);
 }
}
```

The following is the standard mechanism to run a simulation. All the simulation parameters are written in the **totaapp.xml** file. The syntax of this file is strict. All the xml elements must be there in the right order. The current implementation does not support all the reported parameters. Comments have to be removed!

```
<?xml version="1.0"?>
<config>

// This specifies the type of gui to look at the simulator.
// gui.gui2D.Main2D offers an interactive 2D view of the network
<gui_type>
gui.gui2D.Main2D
</gui_type>

// This is the size of the area where the network is contained
// The current implementation does not support 3D networks, just leave z = 0.
<gui_size_x>
700
</gui_size_x>
<gui_size_y>
700
</gui_size_y>
<gui_size_z>
0
</gui_size_z>

// This is the kind of network to be created. Manet, Fixed and Satellite networks are possible
<network_type>
emulator.network.manet.Manet2D
</network_type>

// This is the number of nodes in the network. network_nodes_w is the number of manet nodes.
// network_nodes_x, network_nodes_y, network_nodes_z are used in the fixed network to create
// a rectangular block of nodes. Satellite networks support all the nodes where fixed nodes create
// create the infrastructure and manet nodes connect to that infrastrucutre
<network_nodes_w>
100
</network_nodes_w>
<network_nodes_x>
0
</network_nodes_x>
<network_nodes_y>
0
</network_nodes_y>
<network_nodes_z>
0
</network_nodes_z>

// Network delay and reliability. 0 means no delay, 100 means 100% reliable
<network_delay>
0
</network_delay>
<network_rely>
100
</network_rely>

// Wireless range for manet and satellite networks
<network_wrange>
100
</network_wrange>

// How many messages before saturate network communication buffer
<channel_capacity>
20
</channel_capacity>

// To test energy consumption. Total amount of energy and cost for individual operations
<energy_full>
100
</energy_full>
<energy_receive>
0
</energy_receive>
<energy_send>
```

```
0
</energy_send>

// kind of agent to run on the fixed and manet nodes. Note that here we have
// totaapplication.AppAgent that is the agent we created before

<mas_type_fixed>
null
</mas_type_fixed>
<mas_type_manet>
totaapplication.AppAgent  // IMPORTANT!!!!
</mas_type_manet>

// data used to run batch experiments and collect results
<experiment>
null
</experiment>
<exp_pars>
null
</exp_pars>
<data_collector>
null
</data_collector>
<data_collector_pars>
../CollectedData/prova.txt
</data_collector_pars>
</config>
```

TOTA Meeting Application

Here we report the code of a simplified agent performing the meeting application.

```
package meetingapplication;

import java.util.Vector;

import tota.middleware.TotaMiddleware;
import tota.tuples.TotaTuple;
import tota.tuples.hop.GradientTuple;
import tuplespace.tuples.SensorTuple;
import tuplespace.tuples.Tuple;
import emulator.agent.AbstractAgent;
import emulator.peers.Peer;
import emulator.peers.PeerInterface_ApplicationSide;
import emulator.utils.GenPoint;

public class MeetAgent extends AbstractAgent {

 // this is the group of peer that will try to meet
 private static final String[] group = new String[]{"P0","P1","P2"};
 private static final int speed = 30;

 private PeerInterface_ApplicationSide peer;
 private TotaMiddleware tota;
 private boolean ingroup = false;

 public MeetAgent(PeerInterface_ApplicationSide peer) {
  this.peer = peer;
  tota = new TotaMiddleware(peer);

  for(int i=0; i<group.length;i++)
   if(peer.toString().equals(group[i]))
        ingroup = true;

  System.out.println(peer.getAddress() + " starting...");
 }

 public void step(int time) {
   tota.step(time);
   // nodes that are not in the meeting group run the TOTA middleware, but don't do anything
   if(!ingroup) {
```

```
       peer.setColor(255, 255, 255);
       return;
  }

  // inject the meeting tuple
  if(time == 10) {
   TotaTuple t = new GradientTuple();
   t.setContent("<content=meet>");
   tota.inject(t);
  }

  if(time%50==0) {
   int[] dir = getDirection();
   if(dir != null)
    move(dir[0],dir[1]);
  }
  peer.setColor(255, 0, 255);
}

private int[] getDirection() {
//      read the local tuple space
GradientTuple mt = new GradientTuple();
mt.setContent("<content=meet>");
Vector local = tota.read(mt);

// look in the local tuple space, the meet tuple with the highest hop counter
int maxi = 0;
GradientTuple maxt = (GradientTuple)local.get(0);
for(int i=1; i<local.size(); i++) {
 GradientTuple t = (GradientTuple)local.get(i);
 if(maxt.hop < t.hop)
  maxt = t;
}


if(maxt.hop <= 2) {
 // I am done, all the peers are within one-hop distance
 System.out.println("\n******" +tota.toString()+" MEETING COMPLETE!!!\n");
 return null;
}

System.out.println(tota.toString()+" wants to follow "+maxt.serialize());

// look in the neighbor tuple spaces for neighbor having a lower value of the
// max tuple
GradientTuple tofollow = null;
Vector remote = tota.readOneHop(mt);
for(int i=0; i<remote.size(); i++) {
 GradientTuple t = (GradientTuple)remote.get(i);
 if(t.id.equals(maxt.id) && t.hop < maxt.hop) {
  if(tofollow==null || (tofollow.hop > t.hop))
       tofollow = t;
 }
}

if(tofollow == null)
 return null; // there are no tuples to follow


System.out.println(tota.toString()+" wants to reach "+tofollow.from);

// get the radar tuple
SensorTuple st = new SensorTuple("<sensor=radar><value=*>");
st = (SensorTuple)tota.keyrd(st);
int xy[] = parseRadar(st.value,tofollow.from);
return xy;
}

// this method parse the tuple produced by the low level radar sensor, already installed
// with TOTA to retrieve the position where to go to reduce the gradient
// radar example
// [W][P8@(-72,46,0)][P45@(-100,-11,0)][P55@(-24,1,0)][P74@(71,5,0)][P79@(-69,-42,0)]
private int[] parseRadar(String radar, String peer) {
 int index = radar.indexOf(peer);
 if(index == -1) return null;
 int startindex = index+peer.length()+2;
 int endindex = radar.indexOf(",0)]", startindex);
 radar = radar.substring(startindex, endindex);
```

```
  int comma = radar.indexOf(',');
  int[] xy = new int[2];
  try{
  xy[0] = Integer.parseInt(radar.substring(0,comma).trim());
  xy[1] = Integer.parseInt(radar.substring(comma+1,radar.length()).trim());
  }catch(Exception e){return null;}
  return xy;
 }

 // move toward the indicated direction with a given speed.
 private void move(int dx, int dy){
  Peer p = (Peer) peer.getRealIdentity();
  int[] current = p.getLocation().getCoord();
  GenPoint target = new GenPoint(current[0] + dx, current[1] + dy);
  GenPoint next = p.getLocation().getDirection(target, speed);

  System.out.println(tota.toString()+" is at ("+p.getLocation()+") and moves to ("+next+")");
  p.emu.move(p.toString(), next);
 }

 // interface method empty implementation
 public synchronized void showAgentFrame() {}
 public synchronized void hideAgentFrame() {peer.shutDown();}
 public synchronized Object getRealIdentity() {return this;}
}
```

To run this agent, you have to create another configuration file mainly specifying:

```
<mas_type_manet>
meetingapplication.MeetAgent
</mas_type_manet>
```